**Lab Manual – Structs**

**Important Note:**

**Test 1:** First implement this design using classes. Keep default data members’ access and default inheritance. Run and test your implementation.

**Test 2:** Replace keyword class with struct. Keep default data members’ access and default inheritance. Run and test your implementation.

**a)** Suppose we inherit every class from a parent class Object, as follows:

class Object {

public:

virtual char\* toStr() {return NULL;}

};

class Student : public Object {

private:

int id; char name[25]; float gpa;

public:

// Add code here:

};

class Book : public Object {

private:

char title[100]; int noOfPages; float price;

public:

// Add code here:

};

Here the method toStr() shall convert data of the object into a C-string. For example, consider the following code:

Student s(1, "Ahsan", 2.50);

cout << s.toStr();

Book b("Lord of the Rings", 1000, 50.5);

cout << b.toStr();

The above code shall generate the following output:

**1 Ahsan 2.50**

**Lord of the Rings 1000 50.5**

Now your task is to override the toStr() function in the two child classes i.e. Student and Book.

**b)** Consider a class GenericArray that can store array of objects of any child class of Object. The following code demonstrates use of this class:

int main() {

Student \* ps1 = new Student(1, "Ahsan", 2.50);

Student \* ps2 = new Student(2, "Yasir", 2.75);

GenericArray A(100); // create array of size 100

A.set(0, ps1); // set first index (shallow copy only)

A.set(1, ps2); // set second index

Book \* pb1 = new Book("Lord of the Rings", 1000, 50.5);

Book \* pb2 = new Book("Davinci Code", 700, 40.5);

A.set(0, pb1);

A.set(0, pb2);

for (int j = 0; j < 200; ++j)

cout << A.get(j)🡪toStr() << endl;

return 0;

}

Note that this generic code is not using templates! One advantage of this design choice is savings of memory.

Now give C++ code for the class GenericArray. Include all the required functions. You do not need to write any destructor or deallocation code.